1

Verifiable Binary Lifting

Joe Hendrix, Andrew Kent and Simon Winwood
Galois, Inc
HCSS 2021

© Galois, Inc 2021

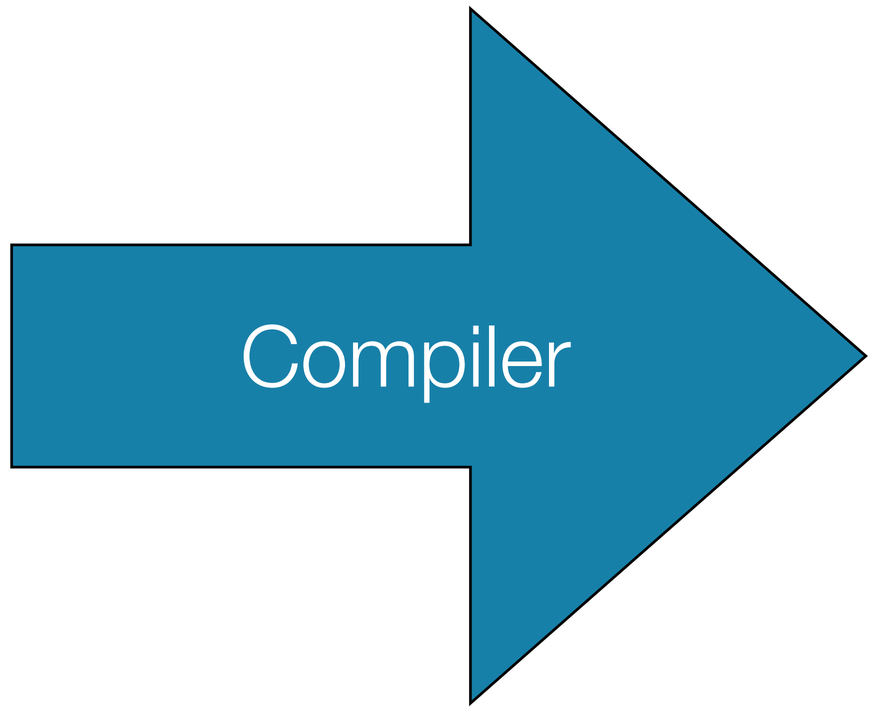
New Applications of Decompilers

© Galois, Inc 2021

What is Decompilation?

- A **compiler** translates code written in a high-level language into a low level language for efficient execution.

```c
uint64_t fib(uint64_t x) {
    if (x <= 1) {
        return x;
    } else {
        return fib(x-1)+fib(x-2);
    }
}
```



```
0000000000201000 fib:
  201000: 55             pushq %rbp
  201001: 4889e5         movq %rsp, %rbp
  201004: 4883ec20       subq $32, %rsp
  201008: 48897df0       movq %rdi, -16(%rbp)
  20100c: 48837df001     cmpq $1, -16(%rbp)
  201011: 0f870d000000   ja   13 <fib+0x24>
  201017: 488b45f0       movq -16(%rbp), %rax
  20101b: 488945f8       movq %rax, -8(%rbp)
  20101f: e934000000     jmp  52 <fib+0x58>
  201024: 488b45f0       movq -16(%rbp), %rax
  201028: 482d01000000   subq $1, %rax
  20102e: 4889c7         movq %rax, %rdi
  201031: e8caffffff     callq -54 <fib>
  201036: 488b4df0       movq -16(%rbp), %rcx
  20103a: 4881e902000000 subq $2, %rcx
  201041: 4889cf         movq %rcx, %rdi
  201044: 488945e8       movq %rax, -24(%rbp)
  201048: e8b3ffffff     callq       -77 <fib>
  20104d: 488b4de8       movq -24(%rbp), %rcx
  201051: 4801c1         addq %rax, %rcx
  201054: 48894df8       movq %rcx, -8(%rbp)
  201058: 488b45f8       movq -8(%rbp), %rax
  20105c: 4883c420       addq $32, %rsp
  201060: 5d             popq %rbp
  201061: c3             retq
```

- A **decompiler** reverses steps in this translation

© Galois, Inc 2021

Who uses Decompilers?

- Decompilers are traditionally used by reverse engineers trying to understand a program.
  - Decompile into a **language understandable by people**.
  - User works with the decompiler to translate code into idiomatic code.
- Without hints or existing source to target, it is generally impossible to recover the original source.
  - Information lost includes all the structure within function bodies such as original control flow structure and local variables.
  - Much more information is lost when compiling with **optimization**.
- More recent programs are aimed at using decompilers for **program transformation** and **repair**.

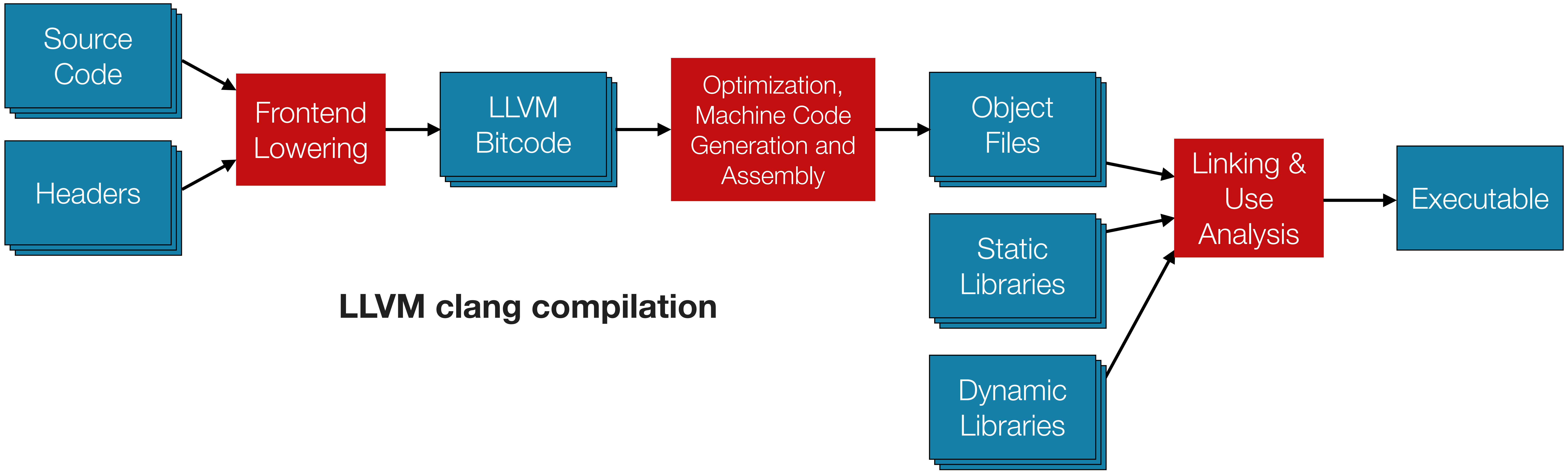
© Galois, Inc 2021

Decompilation for Program Transformation

- Researchers are increasingly looking at using decompilers to transform programs.
  - Patch code with vulnerabilities.
  - Extract functionality from legacy code for use in new applications.
  - Apply new compiler optimizations or insert security checks into legacy applications.
  - Port a program from one platform to another.
- These new applications place greater emphasis on **program correctness** and may have less emphasis on **programmer understanding.**

© Galois, Inc 2021

Compilation Toolchain



LLVM clang compilation

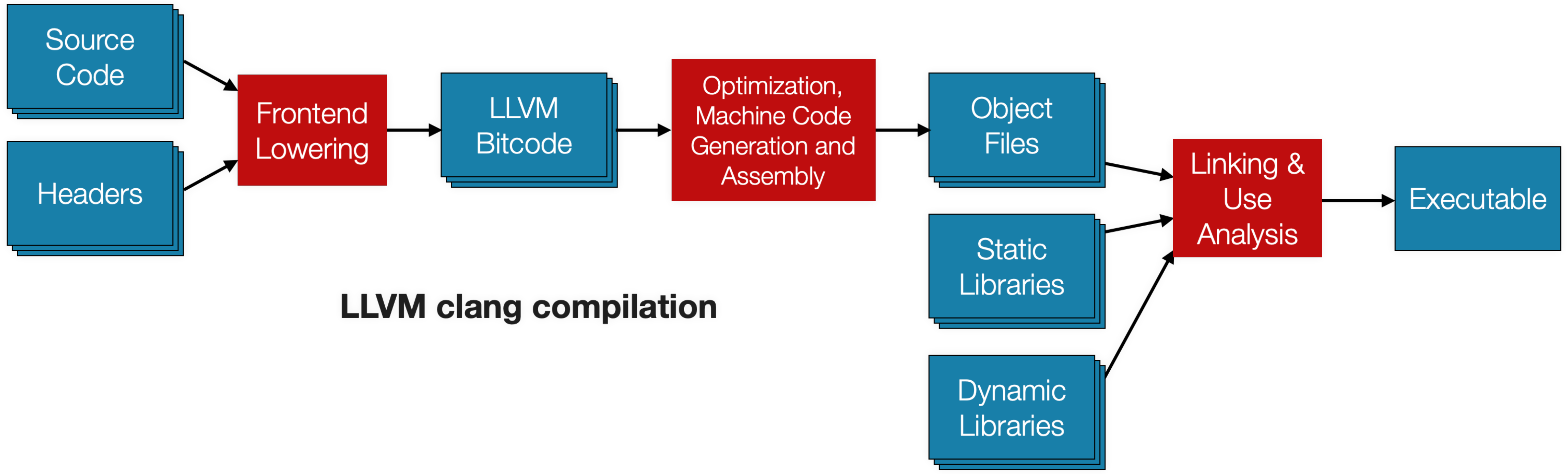
- **Decompilation** needs to reverse these steps.

© Galois, Inc 2021

Recompilation Observations

Recompilation use case differences.

- Sufficient to lift to **compiler IR** or object file representation rather than source.
- **Assured** decompilation is much more important.

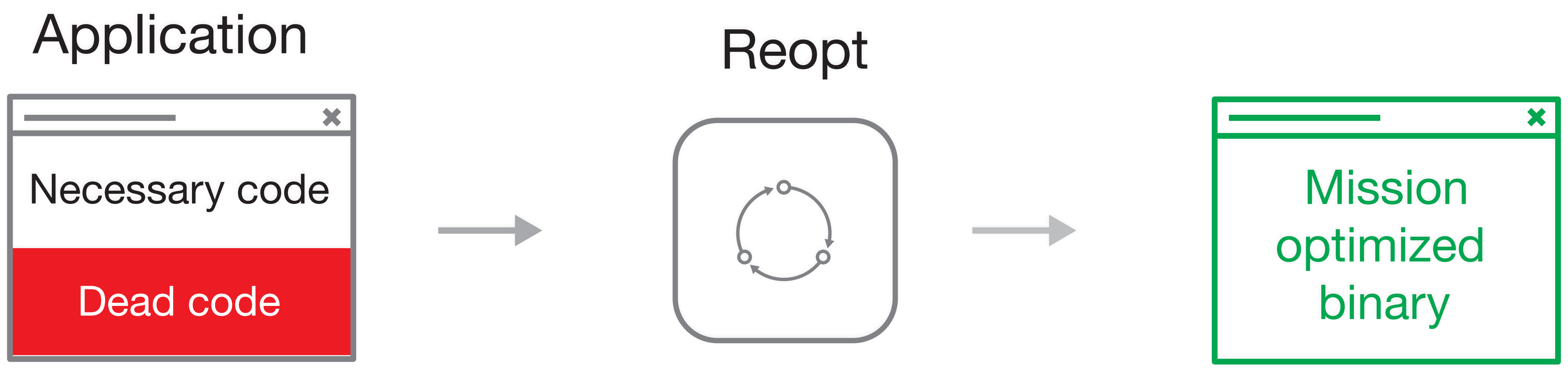


LLVM clang compilation

© Galois, Inc 2021

Program Recompilation

- My talk today is focused on **reopt**, a tool for optimization of compiled executables.



- This can be used for optimization, dead code elimination, and hardening legacy binaries.

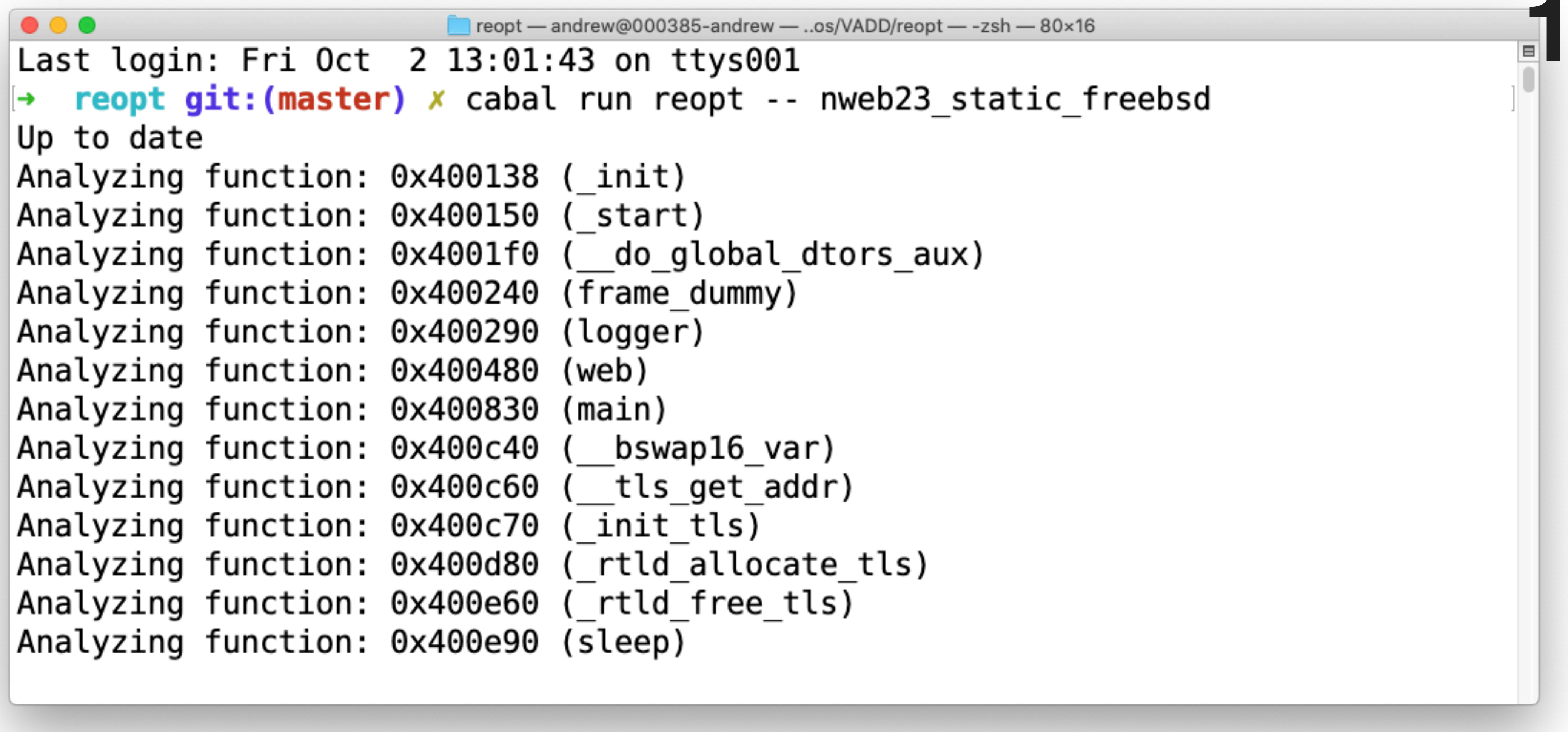
© Galois, Inc 2021

Three Step Process

Three Step Process

1. Decompilation
2. Optimized Compilation
3. Relinking

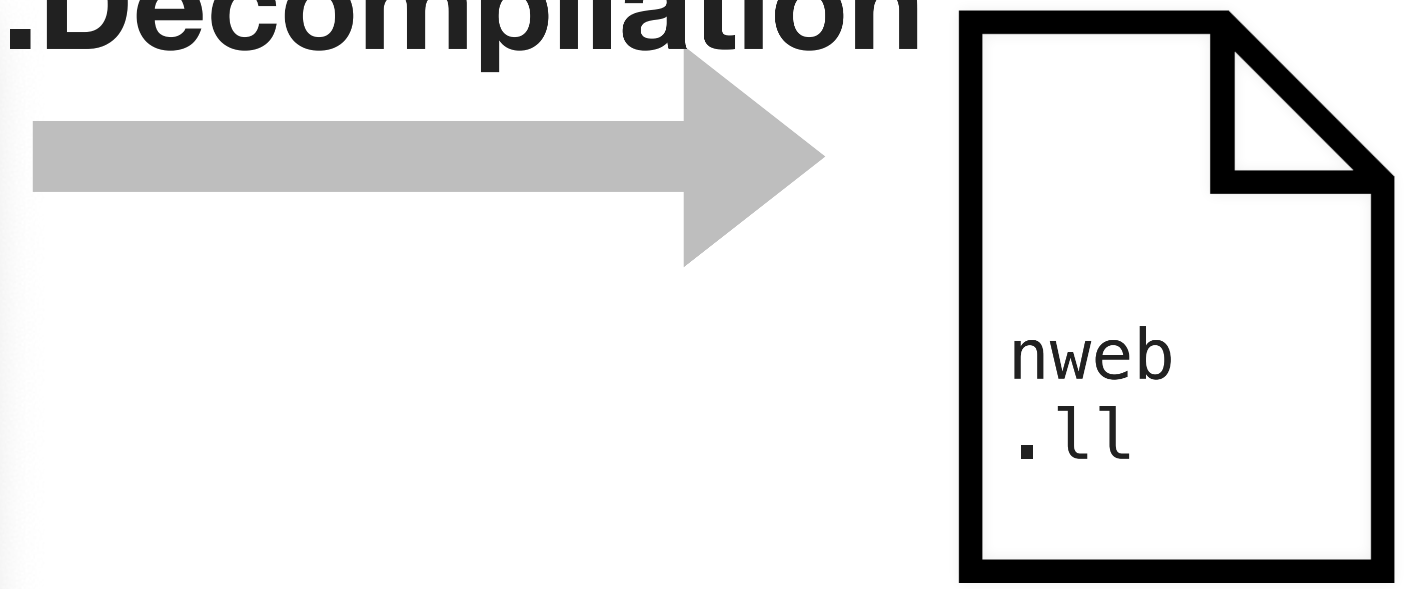
© Galois, Inc 2021

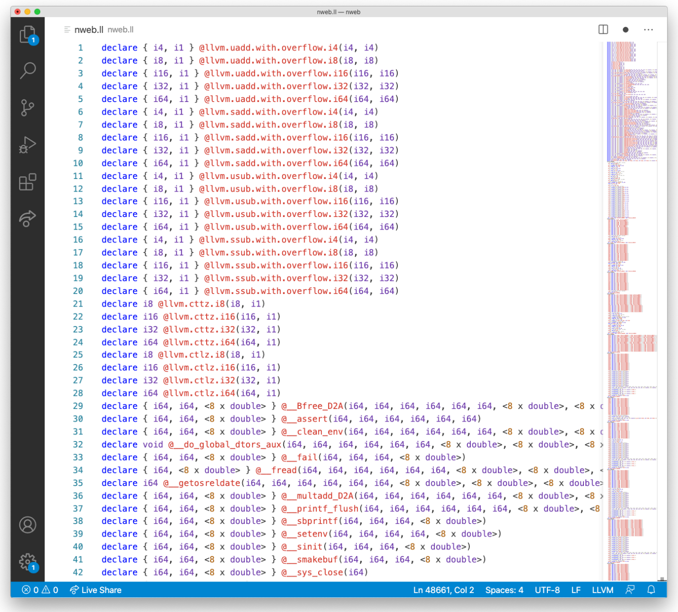


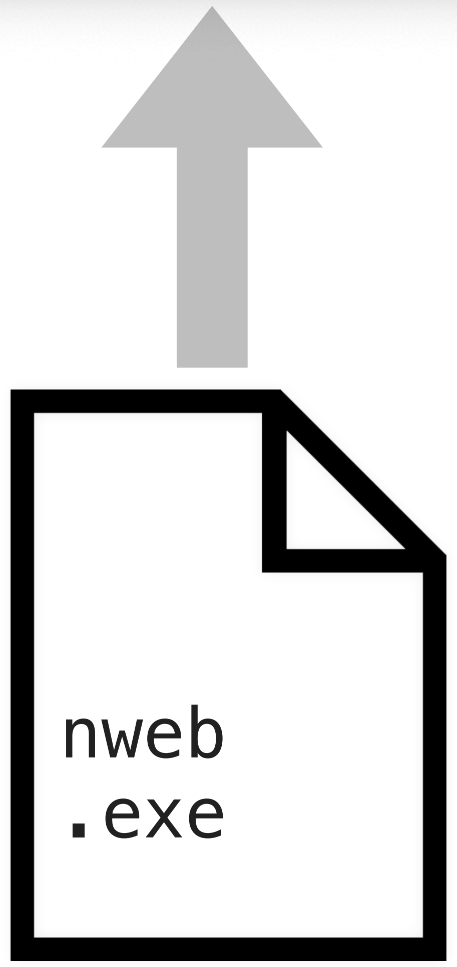
reopt — andrew@000385-andrew — ..os/VADD/reopt — -zsh — 80×16

```
Last login: Fri Oct  2 13:01:43 on ttys001
→  reopt git:(master) ✗ cabal run reopt -- nweb23_static_freebsd
Up to date
Analyzing function: 0x400138 (_init)
Analyzing function: 0x400150 (_start)
Analyzing function: 0x4001f0 (__do_global_dtors_aux)
Analyzing function: 0x400240 (frame_dummy)
Analyzing function: 0x400290 (logger)
Analyzing function: 0x400480 (web)
Analyzing function: 0x400830 (main)
Analyzing function: 0x400c40 (__bswap16_var)
Analyzing function: 0x400c60 (__tls_get_addr)
Analyzing function: 0x400c70 (_init_tls)
Analyzing function: 0x400d80 (_rtld_allocate_tls)
Analyzing function: 0x400e60 (_rtld_free_tls)
Analyzing function: 0x400e90 (sleep)
```

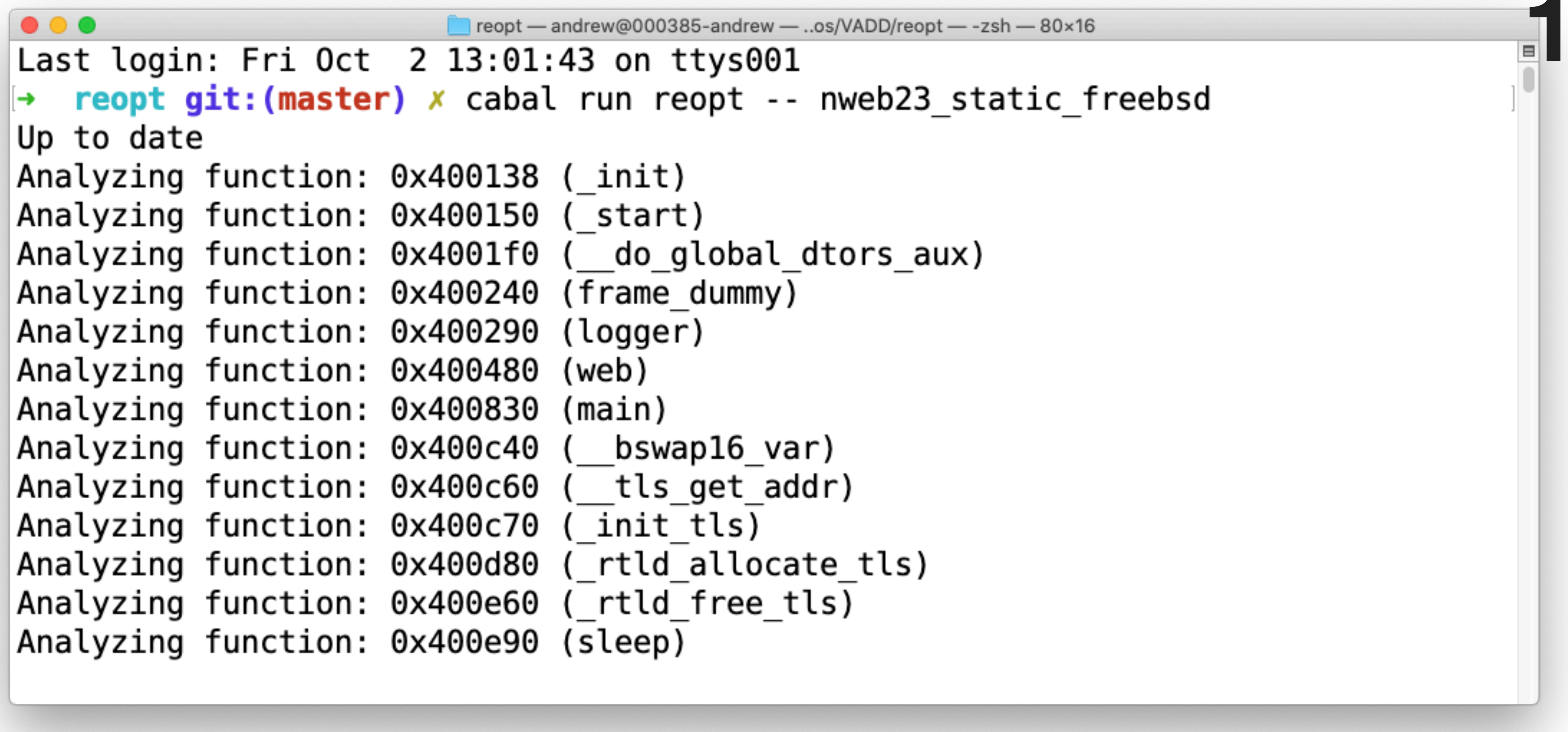
1.Decompilation







© Galois, Inc 2021

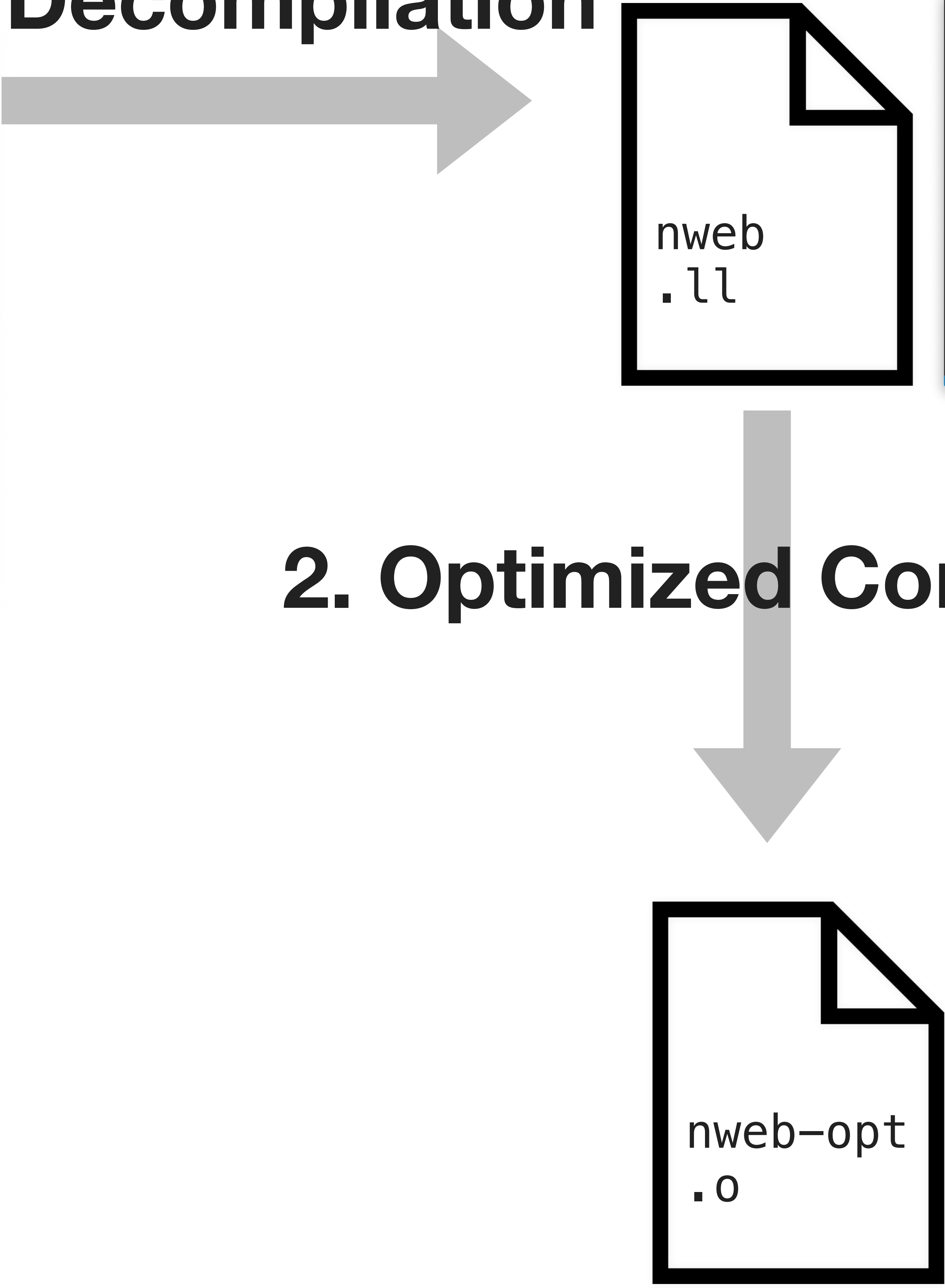


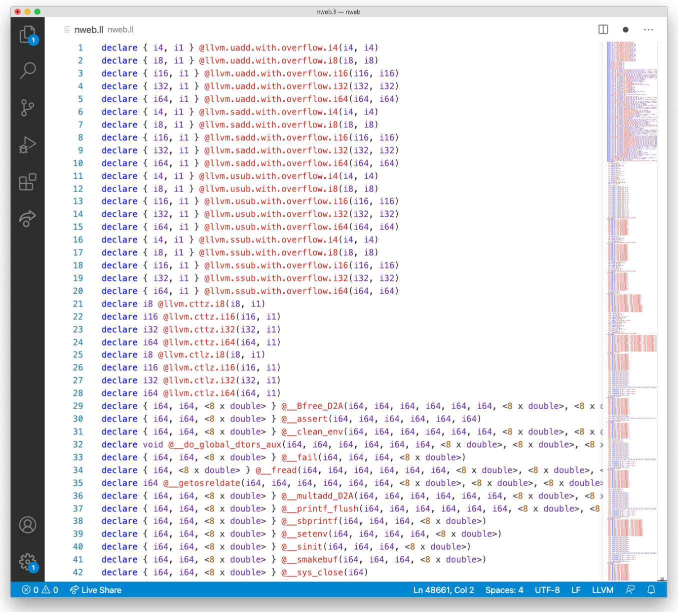
reopt — andrew@000385-andrew — ..os/VADD/reopt — -zsh — 80×16

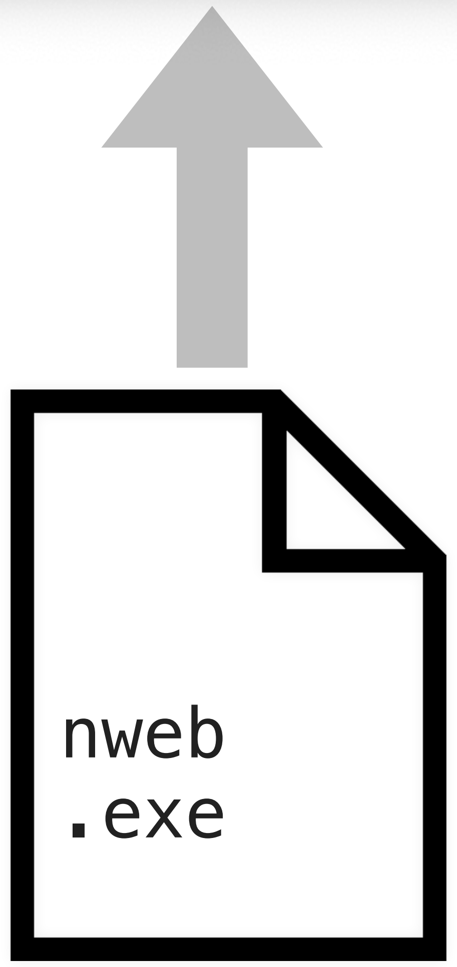
```
Last login: Fri Oct  2 13:01:43 on ttys001
➜  reopt git:(master) ✗ cabal run reopt -- nweb23_static_freebsd
Up to date
Analyzing function: 0x400138 (_init)
Analyzing function: 0x400150 (_start)
Analyzing function: 0x4001f0 (__do_global_dtors_aux)
Analyzing function: 0x400240 (frame_dummy)
Analyzing function: 0x400290 (logger)
Analyzing function: 0x400480 (web)
Analyzing function: 0x400830 (main)
Analyzing function: 0x400c40 (__bswap16_var)
Analyzing function: 0x400c60 (__tls_get_addr)
Analyzing function: 0x400c70 (_init_tls)
Analyzing function: 0x400d80 (_rtld_allocate_tls)
Analyzing function: 0x400e60 (_rtld_free_tls)
Analyzing function: 0x400e90 (sleep)
```

1.Decompilation

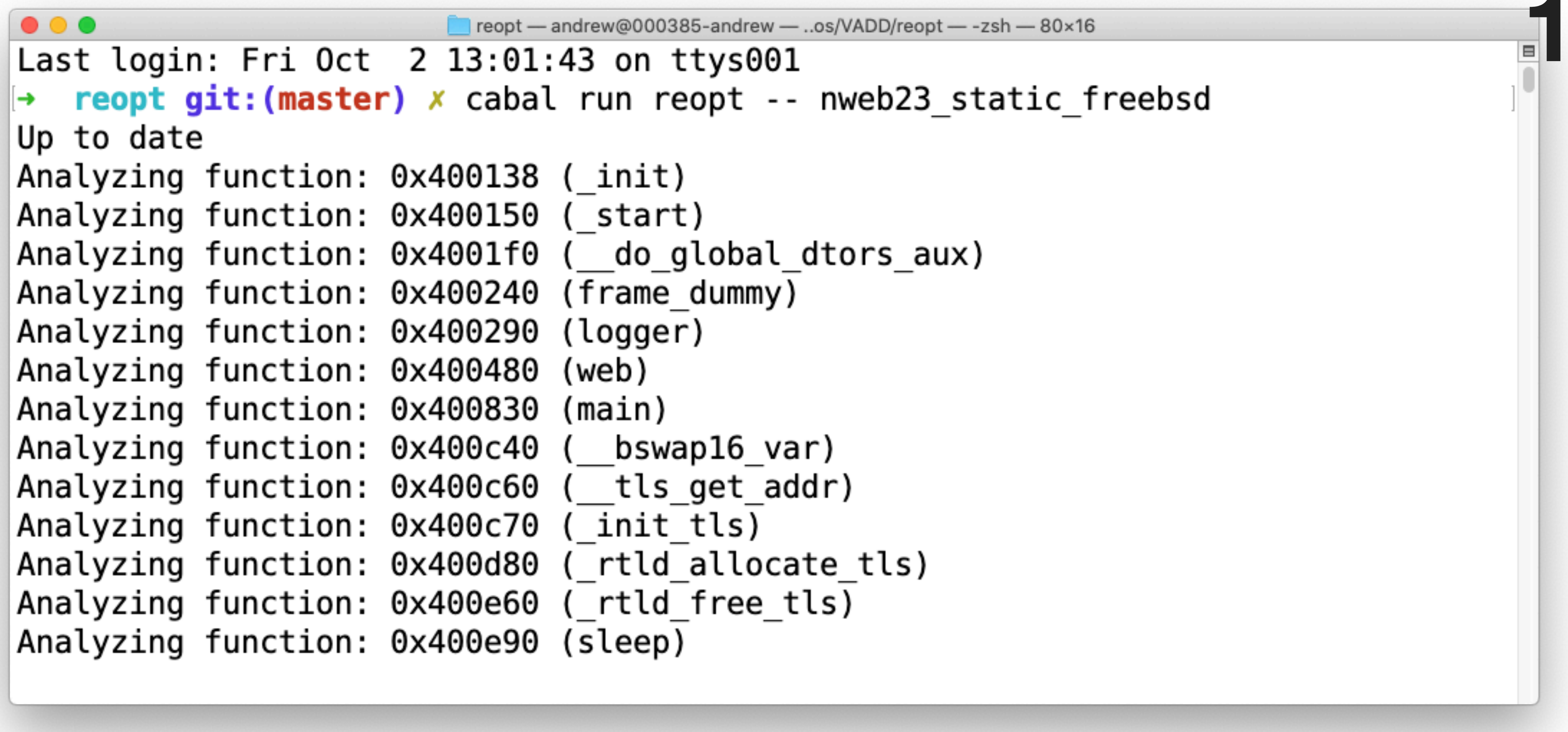
2. Optimized Compilation



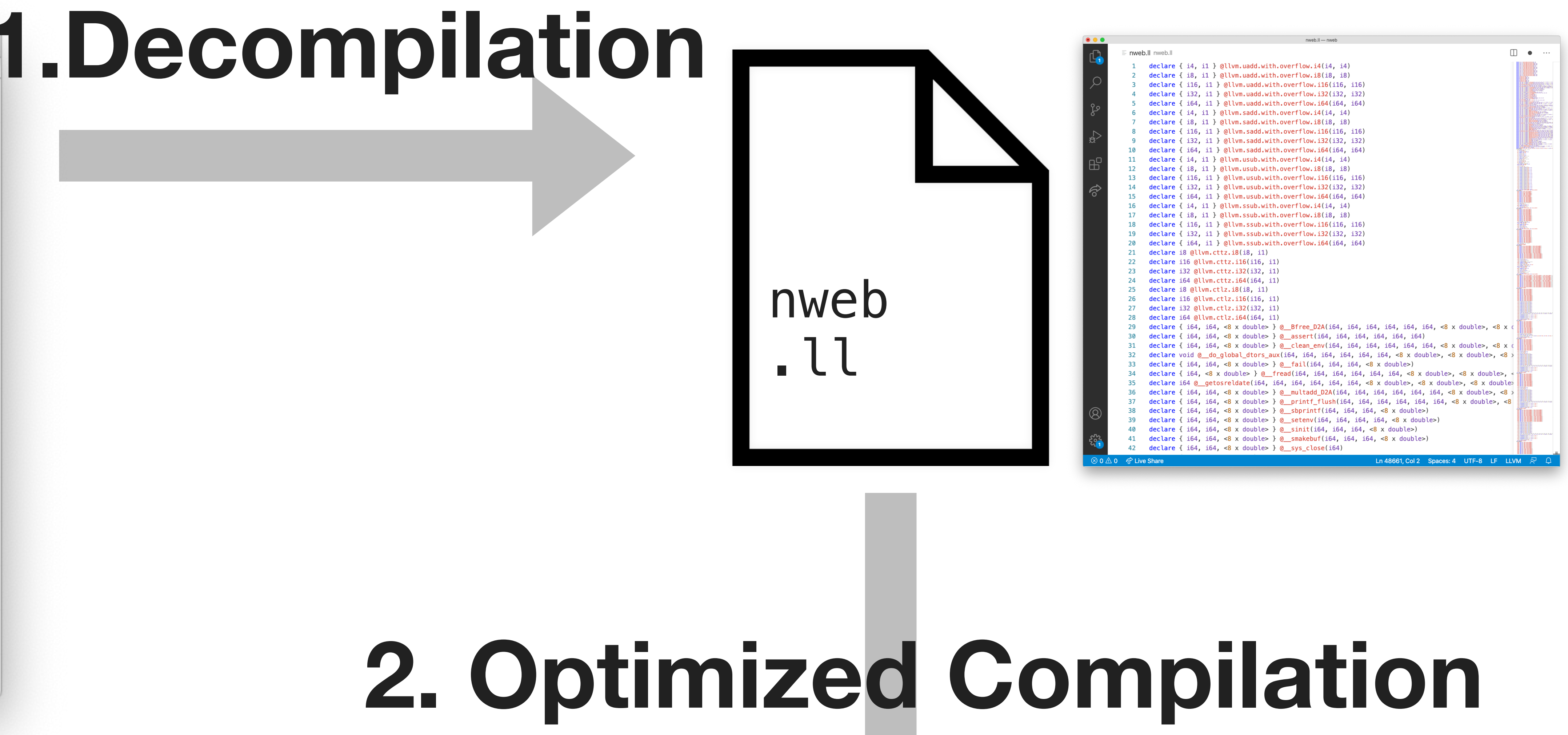




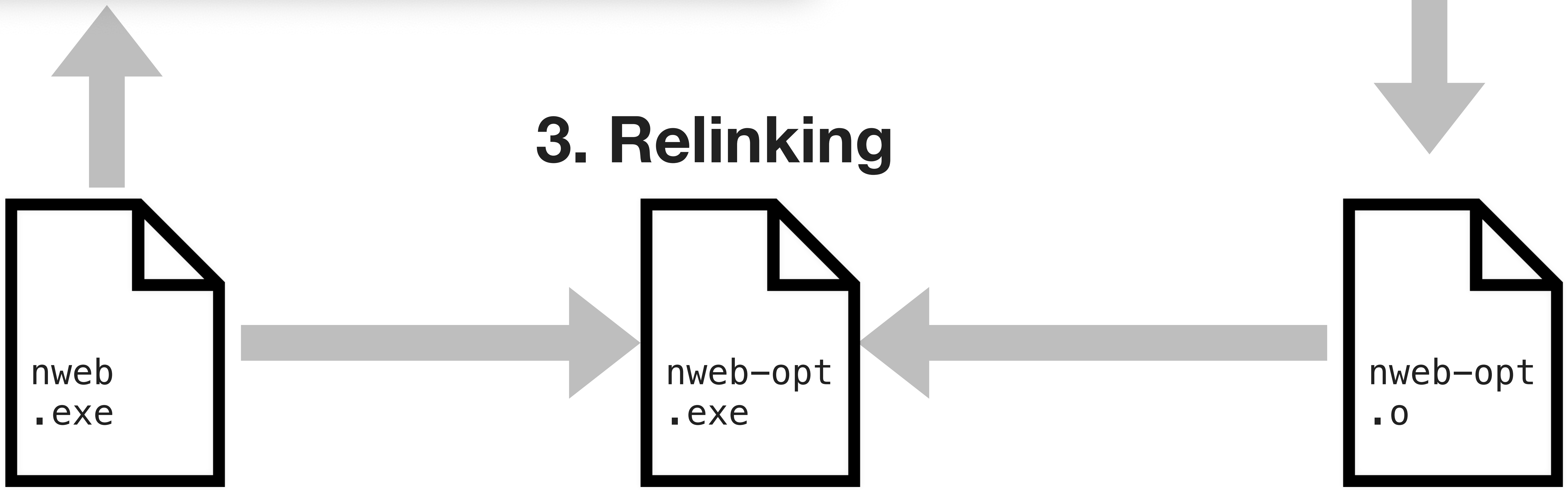
© Galois, Inc 2021


reopt — andrew@000385-andrew — ..os/VADD/reopt — -zsh — 80×16

```
Last login: Fri Oct  2 13:01:43 on ttys001
➜  reopt git:(master) ✗ cabal run reopt -- nweb23_static_freebsd
Up to date
Analyzing function: 0x400138 (_init)
Analyzing function: 0x400150 (_start)
Analyzing function: 0x4001f0 (__do_global_dtors_aux)
Analyzing function: 0x400240 (frame_dummy)
Analyzing function: 0x400290 (logger)
Analyzing function: 0x400480 (web)
Analyzing function: 0x400830 (main)
Analyzing function: 0x400c40 (__bswap16_var)
Analyzing function: 0x400c60 (__tls_get_addr)
Analyzing function: 0x400c70 (_init_tls)
Analyzing function: 0x400d80 (_rtld_allocate_tls)
Analyzing function: 0x400e60 (_rtld_free_tls)
Analyzing function: 0x400e90 (sleep)
```

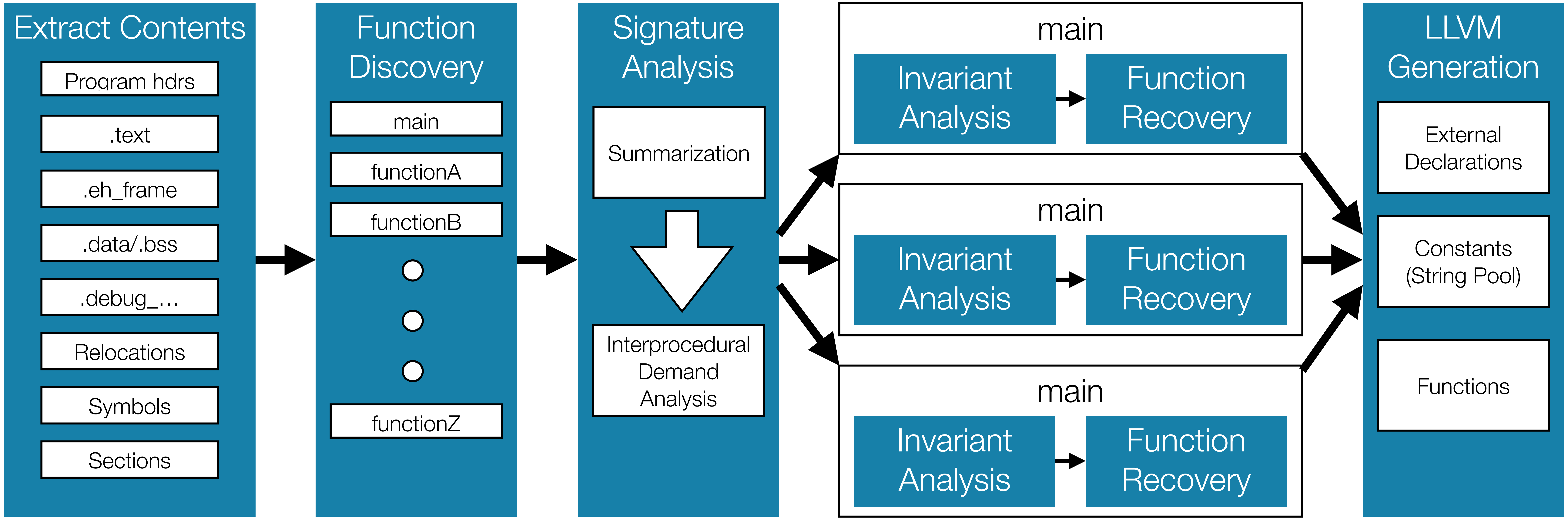

1.Decompilation

2. Optimized Compilation


3. Relinking

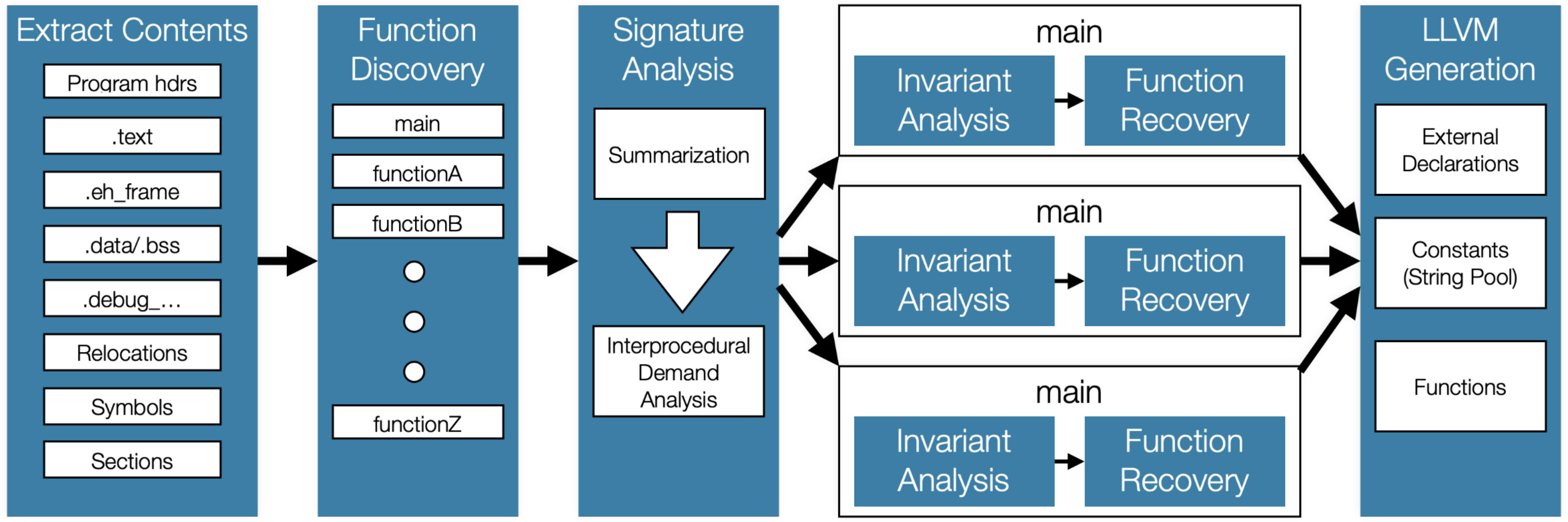
© Galois, Inc 2021

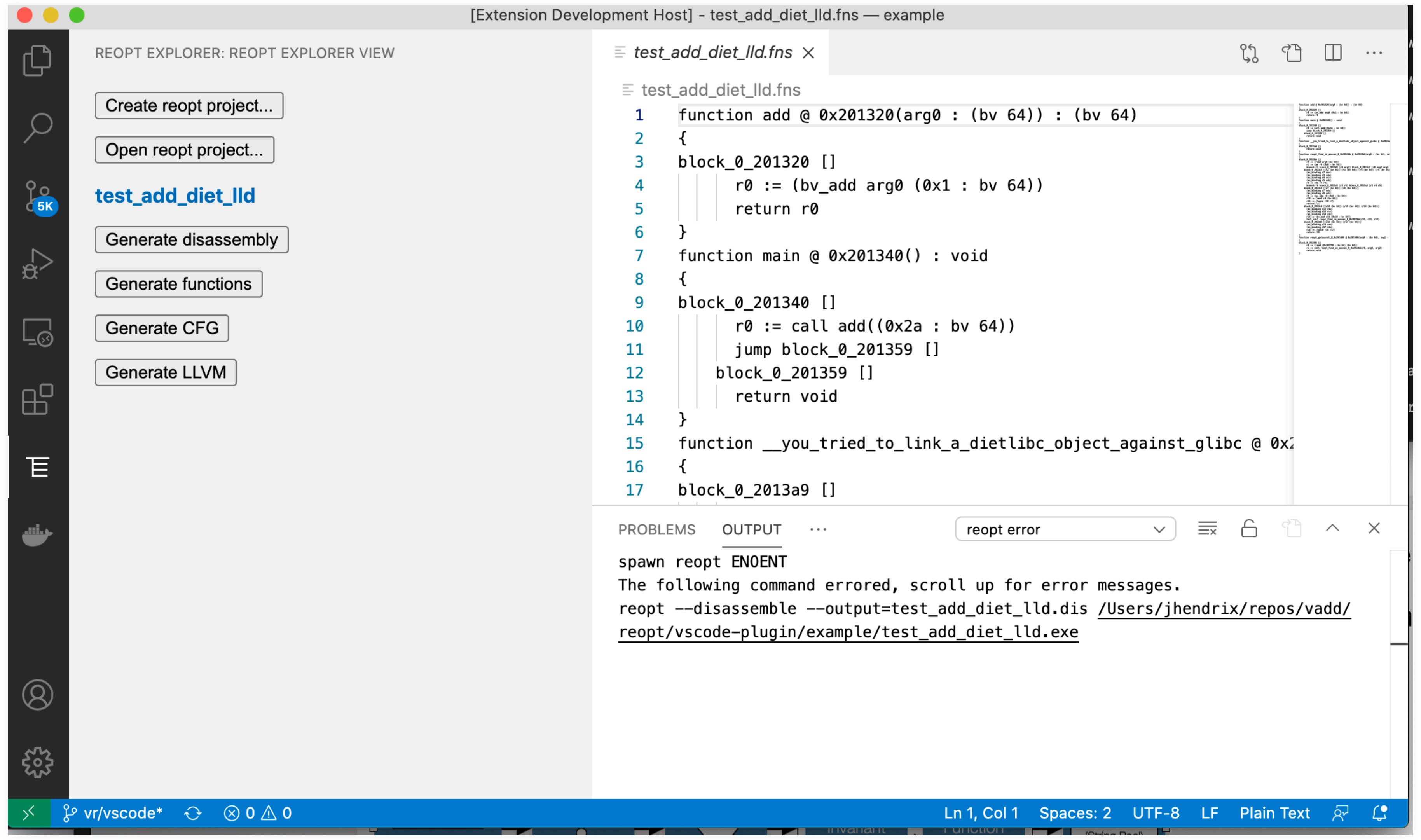
Decompilation Pipeline



Compositionality

- Can export intermediate results at each stage of pipeline.
- Import user information such as additional entry points and function arguments.





test_add_diet_lld.fns

```
function add @ 0x201320(arg0 : (bv 64)) : (bv 64)
{
block_0_201320 []
    r0 := (bv_add arg0 (0x1 : bv 64))
    return r0
}
function main @ 0x201340() : void
{
block_0_201340 []
    r0 := call add((0x2a : bv 64))
    jump block_0_201359 []
  block_0_201359 []
    return void
}
function __you_tried_to_link_a_dietlibc_object_against_glibc @ 0x2
{
block_0_2013a9 []
```

```
spawn reopt ENOENT
The following command errored, scroll up for error messages.
reopt --disassemble --output=test_add_diet_lld.dis /Users/jhendrix/repos/vadd/
reopt/vscode-plugin/example/test_add_diet_lld.exe
```

© Galois, Inc 2021

Verification

© Galois, Inc 2021

Verification Properties

Recompilation Soundness

- Every observable execution in the LLVM should be possible in the machine code program.

$$t \in \text{traces}(P_{\text{LLVM}}) \Rightarrow \exists\, t' \in \text{traces}(P_{\text{MC}}),\ t \equiv t'$$

Verification Soundness

- If a property is true of the raised program, then it should be true of the machine code program.

© Galois, Inc 2021

Observational Equivalence

- Our current notion of equivalence is based on event traces.
- Required events include:
  - Writes to non-stack addresses.
  - Other operations that may raise signals (e.g., divide-by-zero).
  - System calls
- Internally, we make additional equivalence checks for compositional purposes.

© Galois, Inc 2021

Verification Approaches

1. Build a **verified decompiler** using interactive theorem proving.

© Galois, Inc 2021

Verification Approaches

1. Build a **veri** **comp er** using interactive pr .

- Decompilation is an open-ended problem.
- Very complex to implement, and needs continued improvement.

© Galois, Inc 2021

Verification Approaches

1. Build a **verified decompiler** using interactive theorem provers.

- Decompilation is an open-ended problem.
- Very complex to implement, and needs continued improvement.

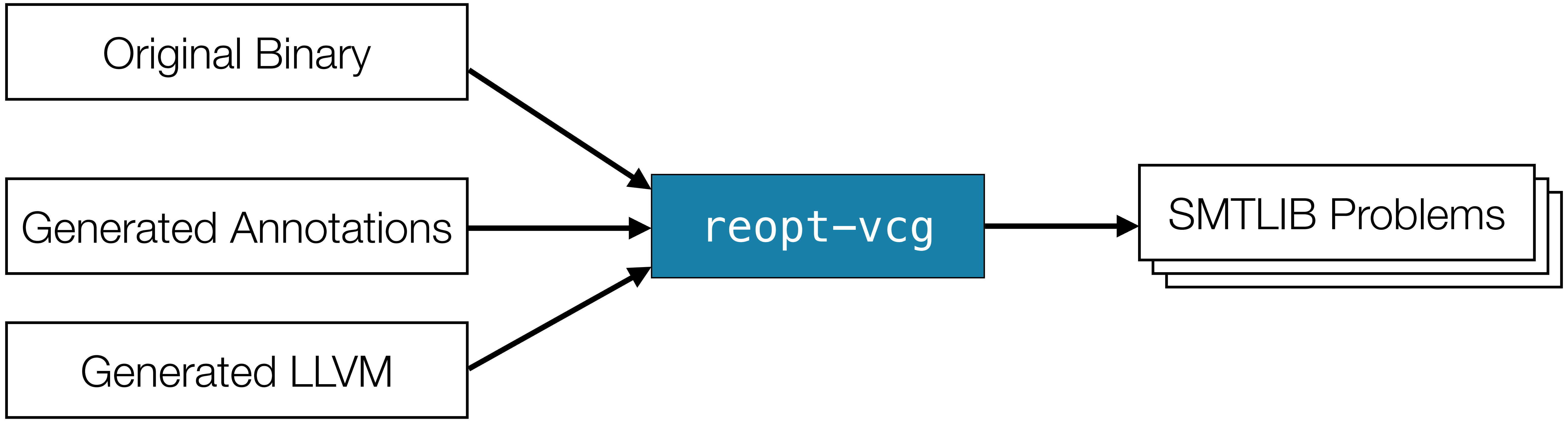
2. Use an automated checker to check the programs are equivalent.

- Program equivalence is ordinarily decidable…
- However, the decompiler output is structurally similar to input binary.
- We have developed a **compositional approach** that checks equivalence of **basic blocks** using SMT solving.

© Galois, Inc 2021

Verification Approach

- We have implemented a verifier based on translation validation.



Correctness claim: If all SMTLIB SAT problems are unsat, then the generated LLVM refines the original binary

© Galois, Inc 2021

Satisfiability Modulo Theories (SMT)

- SMT-based theorem provers can automatically prove theorems involving specific decidable mathematical theories.
- SMT solvers allow decision procedures for different theories to work together.
  - reopt-vcg uses bitvectors, arrays, and uninterpreted functions.

© Galois, Inc 2021

Compositional Proofs

- The key to making automation tractable is to decompose the overall equivalence of programs into many smaller proofs.
- Instead of asking:

  Is LLVM Program P equivalent to machine code program Q?

- We instead ask solvers to answer many questions of the form:

  Is this effect in a LLVM basic block B equivalent to this effect in the machine code?

- For a compositional strategy, we need
  - All the assumptions needed to make the statement true.
  - Check that the assumptions hold when jumping from one block to another.

© Galois, Inc 2021

Compositional Proofs

- Reopt-VCG’s compositional strategy enforces
  - Functions respect the ABI (how arguments are passed, callee-saved registers, etc)
  - The size of each stack frame is bounded to at most a page and all stack accesses are in bounds.
    - Needed to avoid accessing heap memory via stack pointers.
  - Callee saved information is in fact properly saved and not modified during execution of the program.

© Galois, Inc 2021

Getting Reopt

- reopt and reopt-vcg are publicly available under open source libraries.

```
https://github.com/GaloisInc/reopt
```

- You can try it out online through Gitpod, download a Docker image, or use prebuilt binaries.

© Galois, Inc 2021

Thank You

© Galois, Inc 2021